Программирование

Python

Кафедра ИВТ и ПМ

2017

План

Предыдущие темы

IDE

- Jupyter
- Wing

Циклы

- while
- for

составные типы

Списки

Предыдущие темы

- Python компилируемый или интерпретируемый язык?
- Как объявить переменную?

Предыдущие темы

- Python компилируемый или интерпретируемый язык?
- Как объявить переменную?

```
myvariable = 123
```

- Как определить тип переменной?

Предыдущие темы

- Python компилируемый или интерпретируемый язык?
- Как объявить переменную?

```
myvariable = 123
```

- Как определить тип переменной?
  По типу заданного ей значения

Предыдущие темы

- Python компилируемый или интерпретируемый язык?
- Как объявить переменную?

```
myvariable = 123
```

- Как определить тип переменной?
  По типу заданного ей значения
  вызвав функцию: type (имя-переменной)
- Назовите основные простые типы

Предыдущие темы

- Python компилируемый или интерпретируемый язык?
- Как объявить переменную?

```
myvariable = 123
```

- Как определить тип переменной?
  По типу заданного ей значения
  вызвав функцию: type (имя-переменной)
- Назовите основные простые типы
  bool - логический тип
  int - целове число
  float - вещественное число
- Как подключить пакет (модуль)?

Предыдущие темы

- Python компилируемый или интерпретируемый язык?
- Как объявить переменную?

```
myvariable = 123
```

- Как определить тип переменной?
  По типу заданного ей значения
  вызвав функцию: type (имя-переменной)
- Назовите основные простые типы
  bool - логический тип
  int - целове число
  float - вещественное число
- Как подключить пакет (модуль)?

```
import math

import math as M

from math import *
```

Предыдущие темы

- Как выглядит условный оператор?

Предыдущие темы

- Как выглядит условный оператор?

```
if логическое-выражение :
    оператор1
    оператор2
    ...
else:
    операторN
    операторN1
    ...
```

Предыдущие темы

- Как выглядит условный оператор?

```
if логическое-выражение :
    оператор1
    оператор2
    ...
else:
    операторN
    операторN1
    ...
```

- Зачем нужны отступы?
- Где заканчивается условный оператор?
- Сколько отступов рекомендуется использовать для одного блока?

Предыдущие темы

- Как вывести что-то на экран?

Предыдущие темы

- Как вывести что-то на экран?

```
print("текст")
print("скорость равна = ", v, "м/с")
```

- Как прочитать данные с клавиатуры?

Предыдущие темы

- Как вывести что-то на экран?

```
print("текст")
print("скорость равна = ", v, "м/с")
```

- Как прочитать данные с клавиатуры?

```
x = float( input("Введите X ") )
ans = input("Для продолжения введите Y или Yes: ")
```

Outline

Jupyter

Горячие клавиши
Ctrl + **Enter** - Выполнить код в текущей яйчейке
Ctrl + **Shift** - Выполнить код в текущей яйчейке, перейти на следующую (если следующей нет, то она создаётся)
Alt + **Enter** - Вставить яйчейку после текущей

- Весь код и текст содержится в так называемых *блокнотах* (notebooks)
- Блокноты сохраняются автоматически при создании (скорее всего в каталоге текущего плользователя)
- Файл блокнота имеет расширение **ipynb**

Jupyter

- Блокноты состоят из яйчеек
- Яйчейки бывают разные:
  - Яйчейки с текстом. (Markdown)
    Можно использовать LaTeX для описания формул
  - Яейчейки с кодом (code)
- Сохранение notebook.
  Чтобы иметь возможность запускать код блокнота в других средах программирования нужно сохранить его как **py** файл.

Доклад: Как использовать Jupyter (ipython-notebook) на 100%

Демонстрация

Outline

Wing

Демонстрация
Настройки $\rightarrow$ Файлы $\rightarrow$ кодировка по-умолчанию: UTF-8.

В файле исходных кодов добавить:
-*- coding: utf-8 -*-

Ошибки - во вкладке Exceptions

Ввод и вывод во вкладке Debug IO

Циклы

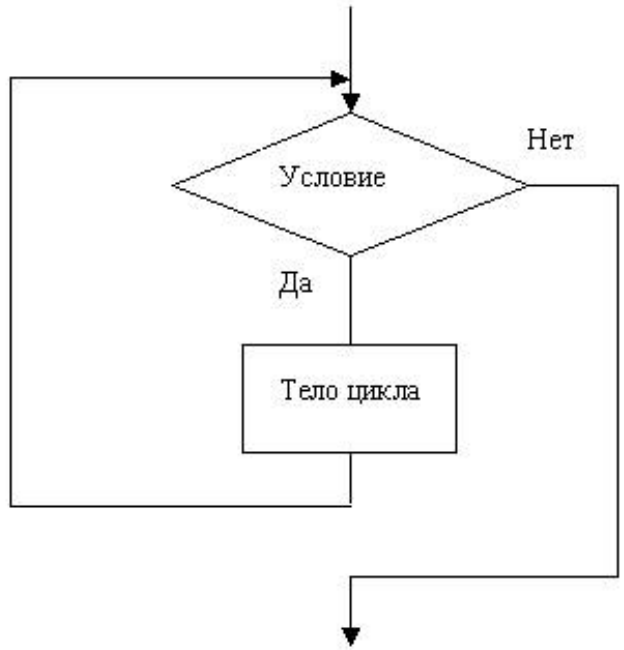
- Цикл с предусловием. while
- Цикл со счётчиком (совместный цикл). for

Outline

while - цикл с предусловием

Короткая форма

```
while условие:    # заголовок цикла
    оператор1     # тело цикла
```



Оператор1 будет выполнятся до тех пор пока условие истинно.

Один шаг цикла и называется **итерацией**.

while - цикл с предусловием

Полная форма

```
while условие :
    оператор1
else:
    оператор2
```

Оператор2 выполнится когда условие станет ложным.

while - цикл с предусловием

Пример. Вычисление факториала числа N

```
N = 6
i = N
F = 1
while i != 1:
    F = F * i
    i = i - 1

print(N,"! =",F)
```

6 ! = 720

Outline

Цикл for

Этот цикл может выполнять перебор элементов в последовательностях.

```
for e in последовательность :   # заголовок цикла
    оператор1                   # тело цикла
    оператор2
    оператор3
```

e - переменаая цикла.

Операторы 1-3 будут выполнятся пока не будут перебраны все элементы в последовательности.
Переменную цикла e объявлять заранее не нужно.

Цикл for

Напечатать все числа из списка:

```
l = [1, 40, -4, 3.14, 0]
for e in l :
    print(e)
```

Что будет напечатано:

1
40
-4
3.14
0

Цикл for

Вышеприведённый вариант (без использования индексации) всегда следует предпочитать варианту с индексацией

```
l = [1, 40, -4, 3.14, 0]
for i in range(len(l)) :
    print(l[i])
```

При прочих равных стоит выбирать то решение, которое короче и не требует дополнительных переменных.

Цикл for

Замечание: Последовательность можно вывести на экран и функцией print():

```
print( [1, 40, -4, 3.14, 0] )
```

Однако, такие последовательности всгда выводятся вместе со сокбками, что может ухудшить восприятие информации:
[1, 40, -4, 3.14, 0]

Цикл for

Переменную цикла можно использовать в выражениях
Напечатать степени двойки:

```
for i in [0, 1, 2, 3, 4, 5] :
    print(2**i)
```

Что будет напечатано:
1
2
4
8
16
32

Циклы

Создание простых последовательностей

Простые последовательности, например 1, 2, ... n, можно не описывать поэлементно (как в предыдущем примере), а создавать с помошью специальной функции.
range(n) - создаёт последовательность из n целых чисел

0..n-1
range(a, b) - создаёт последовательность из b-a целых чисел

a..b-1
Последние число никогда не включается в последовательность.
b должно быть больше чем **a**

Циклы

```
range(5)      # 0, 1, 2, 3, 4
range(0, 5)   # 0, 1, 2, 3, 4

range(3,7)    # 3, 4, 5, 6
range(-3, 1) # -3, -2, -1, 0
```

Циклы

range(a, b, d) - создаёт последовательность от **a** до **b** с шагом **d**

Последние число никогда не включается в последовательность.

```
range(2,10,2)   # 2, 4, 6, 8
range(5,0,-1)   # 5, 4, 3, 2, 1
range(0,25,5)   # 0, 5, 10, 15, 20
```

Циклы

Иногда для анализа работы цикла бывает полезным разобрать его выполнение по шагам.

```
j = 0
k = 0
for i in range(6):
    j = j + i
    k = k + j
```

i	j	k
0	0	0
1	0	0
2	1	1
3	3	4
4	6	10
5	10	20

Взаимозаменяемость циклов for и while

- Цикл while универсален.
- for всегда можно заменить на while
- одноко порой for позволяет решить задачу меньшим количеством кода
- кроме того, использование for иногда даёт выигрыш в производительности.
  Например часто используемая совместно с циклом for функция range создаёт последовательность не целиком, а по одному элементу. Эта функция выдаёт следующий элемент в тот момент когда его нужно записать в переменную цикла.

Когда использовать for, а когда while?

- for подойдёт когда известно число итераций цикла
  Например, число элементов в последовательности
- while можно использовать во всех остальных случаях

Операторы управления циклом

break - прерывает выполнение тела цика и производит бузусловное звершение всего цикла.
Т.е. это оператор безусловного выхода из цикла.
Любые операторы описанные в теле цикла после break не выполняются.

```
while условие :
    оператор1
    оператор2
    break
    оператор3
```

Оператор3 никогда не выполнится. Цикл завершится на первой

итерации
break завершет как цикл while так и for.

Операторы управления циклом

Оператор **break** полезено использовать только совместно с оператором ветвления if.

Поиск индекса элемента в списке

```
i = 0
l = [3, 14, 15, 9, 26, 5]
x = 9  # искомое значение
while i<len(l):
    if l[i] == x:
        break
    else:
        i = i + 1
if i == len(l):
    print("Значение ", x, " не найдено")
else:
    print("Значение '",x,"' находится под порядковым номером
```

Значение ' 9 ' находится под порядковым номером 3

Бесконечные циклы

Оператор **break** позволяет спользовать "бесконечные"циклы:

```
while True :
    оператор1
```

Пример.
Запрашивать у пользователя число натуральное число.

```
print("Введите натуральное число")
while True :
    n = int( input( "n = " ) )
    if n > 0:
        break
    else:
        print("Введите НАТУРАЛЬНОЕ число")
```

Операторы управления циклом

continue - прерывает выполнение текущей итерации цикла и переходит на следующую.

```
while условие :
    оператор1
    оператор2
    continue
    оператор3
```

Оператор3 никогда не выполнится. Цикл завершится

"естетсвенным образом когда условие в заголовке не выполнится.

Операторы управления циклом

Найти действительные квадратные корни элементов последовательности

```
l = [2, 9, 3, -8, -3, -10, 10]
for x in l:
    if x < 0:
        continue
    print("sqrt(",x,") = ",sqrt(x))
```

sqrt(2) = 1.4142135623730951
sqrt(9) = 3.0
sqrt(3) = 1.732050807568772
sqrt(10) = 3.1622776601683795

Операторы управления циклом

continue и **break** - избыточны. Любой цикл можно

организовать без этих операторов. Как обойтись без continue?

Операторы управления циклом

continue и **break** - избыточны. Любой цикл можно

организовать без этих операторов. Как обойтись без continue?

```
for x in l:
    # условие завершения итерации поменялось
    # на условие её проложения
    if x >= 0:
        print("sqrt(",x,") = ",sqrt(x))
```

Операторы управления циклом

Как обойтись без break?

Замечание о объявлении переменной внутри блока

```
if условие :
        x = 10
 else:
        y = 20

  print(x)
  print(y)
```

В чём проблема?

Замечание о объявлении переменной внутри блока

```
if условие :
        x = 10
 else:
        y = 20

  print(x)
  print(y)
```

В чём проблема? Переменная x будет объявлена только если условие выполнится.
Переменная y будет объявлена только если условие **не** выполнится
Однако значения этих переменных выводятся бузесловно, вне зависимости от этого какая переменная была объявлена.
Как решить эту проблему?

Замечание о области видимости переменной

```
x,y = None, None
if условие :
        x = 10
 else:
        y = 20

  print(x)
  print(y)
```

Аналогичная проблема может возникнуть, если перемененная обхявлется внутри любого блока, который может и не выполняться. Наприме в теле цикла.

Составные типы

В программах часть приходится хранить не только отдельные значения, но и целые наборы значений. Причём число таких элементов не всегда может быть известно.
Python для хранения наборов значений используются три категории типов:

- Последовательности (упорядоченный набор значений)
  - Строки (str)
  - Списки (list)
  - Кортежи (tuple)
- Множества (set)
- Отображения (dict)

Списки

В Python нет встроенного типа *массив*.
Вместо них - списки.
Синтаксис обращения к элементам списка похож на синтаксис массивов.
Объявить пустой список:

```
l = []
```

Можно сразу задать значения:

```
l = [ 1, 2, 3 ]
```

Списки

Добавление элементов Списку не нужно задавать начальный

размер потому, что в любой момент можно добавить ещё один элемент в конец.
[42] - это список из одного элемента

```
l = l + [42]   # l = [1, 2, 3, 42]
```

или

```
l.append(42)   # l = [1, 2, 3, 42]
```

Списки

Элементами списка могут быть практически любые типы

```
l = [ 10, "Python", 3.14, True ]
```

В том числе другие списки

```
l2 = [ l, [0,1,2,3] ]
```

Уровень вложенности может быть любым Например можно организовать древовидную структуру

```
tree = [ [[1], 2, [3]], 4, [[5], 6, [7]] ]
```

Как можно представить такой список?

Списки

Некоторые операции со списками

```
l = [10, 20, 30, 42]

n = len(l)  # получить длинну списка
x = l[2]  # доступ к элементам. индексация с 0
# x = 30

z = l[-1]  # доступ к последнему элементу

l2 = [0] * 128  # создание списка из 128 нулей

l2 = [0] * 64 + [1] * 64  # список из 64 нулей и 64 еде

l3 = [0, 1] * 64  # список из чередующихся нулей и едени
```

Списки

Создание списка из последовательности целых чисел

```
l = list( range(0,10) )
```

l = [0,1,2,3,4,5,6,7,8,9]

```
l = list( range(0,10,2) )
```

l = [0,2,4,6,8]

```
l = list( range(10,0,-1) )
```

[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

```
l = list( range(25,0,-5) )
```

[25, 20, 15, 10, 5]

Списки

Создать список из n случайных чисел

```
from random import random
# функция random возвращает псевдослучайное число
# в интервале от 0 до 1
```

Традиционный способ

```
n = 100
l = []
for i in range(n):
    l = l + [random()]
```

Списки

Быстрее и короче. С помощью генератора списка.

```
n = 100
l = [random() for i in range(n)]
```

Списки

Проход по списку.
"Традиционный способ"

```
l = [1,2,3,4]

for i in len(l):
    print(l[i])
```

С помошью совместного цикла

```
l = [1,2,3,4]

for e in l:
    print(e)  # в теле такого цикла нельзя менять e
```

Списки

Если нужно только перебрать элементы одной последовательности то всегда следует использовать перебор безиндексов.
Такой код лаконичнее и лучше воспринимается.
К тому же последовательное обращение к элементам списка просиходит быстрее, чем обращение по номуру элемента.

```
for e in l:
    print(e)  # в теле такого цикла нельзя менять e
```

Доступ к элементам и срезы

```
a = [1, 3, 8, 7]

# К элементам можно обращатся по их номеру от конца списка
a[-1]  # последний, 7
#
a[-4]  # 1
```

Если указан номер несуществуещего элемента, то происходит ошибка времени выполнения

a[20]

IndexError: list index out of range

Доступ к элементам и срезы

Срезы

a[от : до : шаг]

от - если не указан, то от начала списка

до - не включается в срез. по умолчанию - конец списка

шаг - по умолчанию равен 1

`a[:]`

Весь список. от первого элемента до последнего (включая послдений) с шагом 1

`a[::]`

Весь список. от первого элемента до последнего (включая послдений) с шагом 1

3, 8, 7

Доступ к элементам и срезы

```
>>> a[1:]  # со второго элемента до последнего
[3, 8, 7]
>>> a[:3] # с первого элемента до третьего
[1, 3, 8]
>>> a[::2] # с первого и до последнего, с шагом 2
[1, 8]
```

Срезы

```
>>> a = [1, 3, 8, 7]
```

Как получить элементы списка в обратном порядке?

Срезы

```
>>> a = [1, 3, 8, 7]
```

Как получить элементы списка в обратном порядке?

```
>>> a[::-1]
[7, 8, 3, 1]
>>> a[:-2]
[1, 3]
>>> a[-2::-1]
[8, 3, 1]
>>> a[1:4:-1]
[]
```

Двумерный список

```
m = [ [1,2,3], [4,5,6], [7,8,9] ]
for l in m:
    for e in l:
        print(e, end="")
    print()

доступ по индексу
m[1][2]  # 5
```

Двумерный список

Создание двумерного списка (матрицы) n x m

n - число строк

m - число столбцов

```
A = []
for i in range(n):   # цикл по строкам матрицы
    A = A + [ [0]*m ]
```

Ссылки и литература

- Как использовать Jupyter (ipython-notebook) на 100%
  Доклад о Jupyter

Основная литература

- Федоров, Д. Ю. Программирование на языке высокого уровня python : учебное пособие для прикладного бакалавриата Содержит краткое описание языка.
- ru.wikibooks.org/wiki/Python - Викиучебник
- Лутц М. Изучаем Python. 2010. - 1280 с. Содержит подробное описание языка.
- Официальная документация Python3

  `help( имя )`

Ссылки и литература

Ссылка на презентации github.com/VetrovSV/Programming